UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 4

Three Layer Architecture

Date Due: June 7, 2019, 8pm **Total Marks: 95**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- **Make sure your name and student number appear at the top of every document you hand in.** These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- **Assignments must be submitted to Moodle.**
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- **Programming questions must be written in Java.**
- **Non-programming questions must be submitted as either `.txt` or `.pdf` files.** If other file types are used, you will receive a grade of zero for that question.

Version History

- **31/05/2019**: released to students

Preamble

- The objective of this assignment is to restructure assignment 3, and add another I/O option to it. Note that changing code to improve it (but not change its functionality) is called **refactoring**. For projects that last for several years, it is necessary to refactor at least parts of them from time to time.
- The longest class from assignment 3 was `HospitalSystem` which was used to run the system. Also, it was doing several tasks. This assignment will divide up this class using the structure of the 3 layer architecture. In doing this, the project should be structured into the following packages: `commands`, `containers`, `entities`, `startup` and `userInterfaces`.
- All the input and output for assignment 3 was using the console. Input and output using windows can be more pleasing. You are to change the program to give the user the option of console I/O or dialog box I/O.
- The classes from assignment 3, other than the `HospitalSystem` class, should not be changed except to add them to a package. The `HospitalSystem` class will have major changes, although some of it will remain the same. Several parts of the existing `HospitalSystem` class will be moved to new classes.
- It is highly recommended that you use the classes provided in the Assignment 4 folder as your starter code.

Question 0 (2 points):

Purpose: Git for Bonus Marks **(optional)**

As a reminder, you have the opportunity to gain some bonus marks by using git/version control on this assignment. The choice to use git/version control is up to you. This is an incentive program - using git will result in bonus marks, not using git will have no effect on your final grade. You can receive up to 2% extra on your final grade (but you cannot exceed a grade of 100% in the course). Additional details and instructional videos can be found in the folder titled "Git for Bonus Marks" on moodle.

What to Hand In

- Your git-log of Assignment 4

There is a separate submission folder on moodle "Assignment 4 Git Log" for you to upload your git log. You must upload your git log here if you want to bonus marks. Uploading it to the regular Assignment 4 submission folder will result in your git log not being graded.

Evaluation

2pts git logs will be given a grade from 0-2 where:

- 0 indicates that your attempt at using was not good enough to be considered for bonus marks
- 1 indicates an attempt was made but you need to improve the number of commits or the content of your commit messages
- 2 indicates a non-trivial use of git

Question 1 (6 points):

Purpose: Implementing Method Stubs

Degree of Difficulty: Easy

Recall the `HospitalSystem` class from assignment 3. In this class, we used method stubs in place of task 5 (display the empty beds of the ward) and task 7 (release a patient). For this question, your job is to implement these methods and provide a demonstration of them working.

Implement the `displayEmptyBeds()` and `releasePatient()` methods in the provided `HospitalSystem` class. After these are implemented, run the `main()` method of the `HospitalSystem` class and invoke these methods. Copy-and-paste the console input/output into a file titled `a4q1_demonstration.txt`

Once everything is working, rename the class `HospitalSystemA4Q1` before handing it in to moodle. This renaming is strictly to help with marking. When using this class in subsequent questions you can return to the original `HospitalSystem` name.

What to Hand In

- The completed `HospitalSystemA4Q1.java` program with task 5 and task 6 implemented.
- A document titled `a4q1_demonstration.txt` which shows the console input/ouput for task 5 and task 7.

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

2 pts For correctly implementing task 5.

2 pts For correctly implementing task 7.

2 pts For `a4q1_demonstration.txt`.

Question 2 (54 points):

Purpose: Practising refactoring using the `HospitalSystem` class.

Degree of Difficulty: Moderate

The present version of the `HospitalSystem` class has a method to handle each of the tasks. In general, each method does the following things:

- Obtains user input for the data needed for the task
- Checks the data for validity
- Invokes the methods from other classes to do the task
- Handles the result of the task

Any errors are handled by throwing an exception where the error is detected and catching it in the `main` method of the `HospitalSystem` class to issue the error message.

For this question, your job is to refactor the task's in the `HospitalSystem` class. Ideally, we would refactor each task but for this assignment we will only refactor four tasks (details below). The remaining tasks can be left where they are (with a few changes) or can be converted to classes, but all 8 tasks should work correctly when invoking `HospitalSystem.main()`.

(a) (20 points) Create the following new classes to refactor tasks 3, 4, 5 and 8 from the `HospitalSystem` class:

- `NewDoctor` (task 3 from assignment 3)
- `AssignDoctor` (task 4 from assignment 3)
- `EmptyBeds` (task 5 from assignment 3)
- `DropDoctor` (task 8 from assignment 3)

These new classes should extend the provided `CommandStatus` class. The `HospitalSystem` class will still obtain the data from the user and handle the result, but the new classes will do the rest of the task (check the data for validity, and invoke methods in other classes to do the task).

Additional information on `CommandStatus`: Each of the above classes will extend this provided class. Any errors that occur in the carrying out of a task should set the successful field to false, and record an appropriate error message in the errorMessage field. The successful completion of a task should set the successful field to true. Also, if any information is to be returned to the invoking method, the usual return of a function is not to be used. Instead, the information should be placed in appropriate fields of the object for the task, and accessors provided for the client to obtain the information. This is done so that it is possible to delay the invocation of a task and/or delay the accessing of the results of the invocation. It also allows the task to be executed remotely, by sending the task to a remote location and later receiving it back to retrieve the results. We will not use either of these properties, but they are useful properties of tasks.

(b) (20 points) When the tasks are put into separate classes, most of these classes need to access the `Patient` map, the `Doctor` map, and the `Ward`. In order to provide this access but prevent more than one instance of any of these entities, you are to use the **Singleton Pattern** to implement them. Therefore the following three singleton pattern classes must be created:

- `PatientMapAccess`
  - `dictionary`: `TreeMap<Integer, Patient>`
  - `DoctorMapAccess()`
  - `dictionary()`: `TreeMap<Integer, Patient>` returns the dictionary containing all patients known to the system

- `DoctorMapAccess`
  - `dictionary`: `TreeMap<String, Doctor>`
  - `DoctorMapAccess()`
  - `dictionary()`: `TreeMap<String, Doctor>` returns the dictionary containing all doctors known to the system
- `WardAccess`
  - `ward`: `Ward`
  - `WardAccess()`
  - `initialize(name: String, minBedLabel: int, maxBedLabel: int)` creates an instance of a Ward for the hospital system. If the Ward has already been initialized this should throw and error.
  - `ward()`: `Ward` returns the ward known to the hospital system. If the Ward has not been initialized this should throw an error.

The dictionaries required for `PatientMapAccess` and `DoctorMapAccess` should initially be empty. The classes that need these data structures should now use `static` methods to access the them.

(c) (14 points) You will need to include internal documentation for all of the new classes. Proper internal documentation includes:

- A comment just before the class header that gives an overview of the class. For example, if the class models an entity, the comment might state what entity the class models and specify the key features. Alternatively, if the class serves as a container, state what type of items the container stores, and how these items are accessed. If the class is an interface, state what it provides an interface for and whether there is anything special about the interface. Finally, if it has a control function, what is it doing and controlling? Recall that comments for a class appear before the class and begin with `/**`.
- A comment just before each instance variable stating what is stored in the field, again beginning with `/**`
- A comment before each constructor and method stating what it does. Note that the comment only gives what the routine does, not how it does it. If it isn?t obvious from the code how it accomplishes its goal, comments on how it is done belong in the body of the method.
- Single line comments as needed to improve the readability of your code. Over-use of single-line comments will result in a deduction of marks.
- Use descriptive variable names and method names.
- Include good use of white space, especially reasonable indentation to improve the readability of your code.
- When testing, hard-code all of the data. Minimize console input and output by only reporting errors(These are the same principles we applied in CMPT 145).

What to Hand In

- A file titled `A4Q2.jar` of your complete system.
- A zip folder titled `A4Q2.zip` that contains:
  - `HospitalSystem.java`
  - `NewDoctor.java`
  - `AssignDoctor.java`
  - `EmptyBeds.java`

- `DropDoctor.java`
- `PatientMapAccess.java`
- `DoctorMapAccess.java`
- `WardAccess.java`

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

20 pts For the command classes (5 pts per class)

20 pts For the singleton classes (2 pts per required attributed/method)

14 pts For internal documentation of new classes (2 pts per class)

Question 3 (25 points):

Purpose: Practising User Input/Output with Dialog Boxes

Degree of Difficulty: Moderate

The second part of the assignment is to change the input and output (I/O) so that the user has the choice of doing I/O via the console or by dialog boxes. A new instance variable should be added to the `HospitalSystem` class called `ioInterface` that has type `InputOutputInterface`.

(a) (21 points) At the start of the project, a dialog box should be used to query the user as to which type of I/O is to be used. Based on the selection, this new instance variable will either be initialized with a `ConsoleIO` object or `DialogIO` object (both described below) which will allow the requested type of I/O to be used. Recall, a variable can be declared as an interface type. But it must be initialized to an object that implements that interface before it is used. Doing this allows us to utilize polymorphism to achieve a different behaviour throughout our system. This part of the assignment should **NOT** be implemented by having two versions of the methods of HospitalSystem. Instead, you are given an interface, `InputOutputInterface`, with the operations needed for I/O. This interface will have two implementations:

- `ConsoleIO` which uses console input and output to implement the methods.
- `DialogIO` which uses dialog boxes to implement the methods. This class can extend the provided `AbstractDialogIO` class which has the `readChoice` method implemented for you.

(b) (4 points) You will need to include internal documentation for all of the new classes. Proper internal documentation includes:

- A comment just before the class header that gives an overview of the class. For example, if the class models an entity, the comment might state what entity the class models and specify the key features. Alternatively, if the class serves as a container, state what type of items the container stores, and how these items are accessed. If the class is an interface, state what it provides an interface for and whether there is anything special about the interface. Finally, if it has a control function, what is it doing and controlling? Recall that comments for a class appear before the class and begin with `/**`.
- A comment just before each instance variable stating what is stored in the field, again beginning with `/**`
- A comment before each constructor and method stating what it does. Note that the comment only gives what the routine does, not how it does it. If it isn?t obvious from the code how it accomplishes its goal, comments on how it is done belong in the body of the method.
- Single line comments as needed to improve the readability of your code. Over-use of single-line comments will result in a deduction of marks.
- Use descriptive variable names and method names.
- Include good use of white space, especially reasonable indentation to improve the readability of your code.
- When testing, hard-code all of the data. Minimize console input and output by only reporting errors(These are the same principles we applied in CMPT 145).

What to Hand In

- A file titled `A4Q3.jar` of your complete system.
- A zip folder titled `A4Q3.zip` that contains:
  - `HospitalSystem.java`
  - `ConsoleIO.java`

- `DialogIO.java`

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

5 pts For correctly modifying the `HospitalSystem` class to allow for user selected I/O

8 pts For correctly implementing the `ConsoleIO` class (2 pts per method)

8 pts For correctly implementing the `DialogIO` class (2 pts per method)

4 pts For internal documentation of new classes (2 pts per class)

Question 4 (10 points):

Purpose: Practising External Documentation

Degree of Difficulty: Easy

Once you have completed the questions above, create a file called `A4_documentation.pdf` that includes all the internal documentation for the refactored system. External documentation should include:

- A description of how to run your system. What class should be invoked, what method?
- Now that we have the beginning of a complete java application, you should include a demonstration of your system running. This can be achieved by copy-and-pasting (or screen-grabbing) your console input/output that shows you invoking each task.
- The status of your assignment. What is working and what is not working? What is tested and what is not tested? If it is only partially working, the previous point should have described how to run that part or parts that work. For the part or parts not working, describe how close they are to working. For example, some of the alternatives for how close to working are (i) nothing done; (ii) designed but no code; (iii) designed and part of the code; (iv) designed and all the code but anticipate many faults; or (v) designed and all the code but with a few faults; (vi) working perfectly and thoroughly tested.
- Maintenance manual. This is information for the person or persons who must keep your system running, fix any faults, and do any upgrades. The reader of the maintenance manual is expected to have the same background and experience as yourself, but of course not having developed your system. This part of the documentation will vary from assignment to assignment. For this assignment, it is sufficient to include a UML class diagram showing all the features of each class, and the relationships (inheritance, uses and aggregation) amongst the classes (if any).
- UML diagrams should be generated with a program like UMLet or an online tool, such as `https://www.gliffy.com/uses/uml-software/` or `https://www.draw.io/`. Make sure that you use the correct arrows and arrowheads. Incorrect arrowheads will lose several marks. You can also draw out a UML diagram by hand and scan it. Just make sure it is easily readable.

What to Hand In

- A file titled `A4_documentation.pdf`

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

10 pts For including all the required external documentation.

Additional information

Commenting and Exceptions: When writing these classes, be sure to properly document each method, including `@param` and `@return` comments. Also, if a method has a precondition, specify the precondition in a `@precond` comment, and throw a runtime exception if it is not satisfied. Be sure to include a meaningful error message in the String parameter for the exception constructor.

Levels of Protection: In keeping with the principle of information hiding, the instance variables of a class should be private unless there is a very good reason to make them visible. Methods should be public unless there is a very good reason to hide them.